Simulation Modelling Practice and Theory 30 (2013) 74–89

Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Integration of symbolic modelling within an equation based flowsheet package for steam pyrolysis

Marco W.M. van Goethem [a,*], Peter J.T. Verheijen [b]

[a] Technip Benelux B.V., Pyrotec Division, Boerhaavelaan 31, P.O. Box 86, 2700 AB, Zoetermeer, The Netherlands
[b] Biotechnology Department, Delft University of Technology, Julianalaan 67, 2628 BC Delft, The Netherlands

ARTICLE INFO

Article history:
Received 15 May 2011
Received in revised form 29 August 2012
Accepted 30 August 2012
Available online 11 October 2012

Keywords:
Modelling language
Symbolic mathematics
Equation-based
Steam cracking
Pyrolysis

ABSTRACT

This paper describes the experience gained from the integration of symbolic modelling within an existing equation-based flowsheet package, SPYRO® Suite 7. A demand for more flexibility led to the integration of a symbolic model definition module into the existing program. We elaborate on our motivations and choices made to our decision for the gPROMS language. The existing program is described, and in some detail the functionalities to minimise the modelling errors and to enhance the detection of them. The gPROMS language itself, the evolution of it and the selected subset are described. Additionally, new attributes of the language are described: spline construction, intermediate graphical results, and definition of a large number of simulations/optimisations. The symbolic model definition module is demonstrated on the determination of an attainable region and the thermal cracking of ethane. The symbolic modelling module proved to be cost effective and improved the quality of the main engine.

1. Introduction

The application of computer-aided process engineering in the industry was boosted in the late fifties by the introduction of the third generation programming languages, such as FORTRAN. The simulation activities were focused in the beginning on individual pieces of equipment, for improving the knowledge and the operation and later on systems consisting of multiple units of equipment. In the eighties flowsheet simulators of unit operations were a common tool for all process engineers. In the nineties the dynamic simulators abandoned the research and development stage and yielded equation-based simulators, such as ASCEND [1], DYMOLA [7], SPEEDUP [24], and gPROMS [23]. From the late nineties until current date we are waiting for the physicochemical simulators, such as MODEL.LA [4] or MODKIT [5] to leave the development phase. These simulators have not penetrated into the industrial practice due to a lack of commercial implementation of these tools [16]. Therefore they do not leave the development phase.

In this work we are dealing with a simulation tool, SPYRO® for the steam pyrolysis process that was introduced in the late seventies [9] and has evolved over the years such as other simulation tools. It started as a simulation program to describe the physical and chemical phenomena solely inside the radiant coil. It has evolved both on physicochemical as on application areas. Improvements in the kinetic schemes were made and models for other unit operations were added to become a full sequential modular simulator for the steam pyrolysis process. In the late nineties it became evident that the sequential modular paradigm on which the program is founded reached its limits for functional expansion. The increasing need for more

* Corresponding author. Tel.: +31 630373069.
E-mail address: mvangoethem@technip.com (M.W.M. van Goethem).

1569-190X/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.simpat.2012.08.005

Nomenclature

A	area (m^2)
C	molar concentration (mol m^{-3})
F	molar flow rate (mol s^{-1})
F_0	molar flow rate of the feed (mol s^{-1})
h	height (m)
K	product removal function (m^{-3})
k	reaction constant (s^{-1} or m^3 mol^{-1} s^{-1})
L	feed distribution function (m^{-3})
$M(V,v)$	mixing kernel (amount mixed from location V to v) (m^{-6})
Mw	molecular mass (kg mol^{-1})
n	total number of components (–)
P	pressure (Pa)
r	radius (m)
r_{RH}	ratio radius-height (m m^{-1})
R	net production rate of a component (mol m^{-3} s^{-1})
R_g	universal gas constant (J mol^{-1} K^{-1})
T	temperature (K)
V, v	volume (m^3)
V_t	total volume (m^3)
x_{wt}	wet feed mass fractions (–)
$x_{wt,HC}$	hydrocarbon feed mass fractions (–)

Subscript

k	identifier for the kth component j,m index for components

Greek symbols

ϕ_m	mass flow rate (kg s^{-1})
ρ	density (kg m^{-3})
τ	residence time (s)

Acronyms

HC	hydrocarbon
SDR	steam dilution ratio
SMD	symbolic model definition
OCFE	orthogonal collocation on finite elements

flexibility of the modelling capabilities led to the development of an equation-based version, SPYRO,[1] a more elaborated review of this is given by Barendregt et al. [2]. This increase in modelling capabilities is two fold. The first is the removal of certain structural limitations, such as a fixed predefined flow sheet for the simulation [35], enabling novel process designs. The second is the incorporation of mathematical optimisation capabilities [36]. The latter yielded a modelling module for the definition of objective functions, inequality constraints and additional equality constraints (besides the predefined models). It proved fortuitous that the implementation of a module, where the optimisation problem could be formulated symbolically, provided additional flexibility. The features of this modelling module were limited. It is only applicable for optimisation and not for simulation, the modelling language used had a flat structure, no sub-models, no inheritance, and no structured equations (FOR-loop, IF-statement, CASE-statement). This initiated the integration of an advanced symbolic modelling module in an existing equation-based flow sheet package. This work can be generalised to incorporate symbolic modelling in any dedicated simulation/optimisation program. The aim of this paper is to present the experience gained by the integration of a symbolic module in an existing equation-based flowsheet program.

In the remaining part of this introduction we will elaborate on the motivation and choices made. Thereafter we will describe the equation-based simulation/optimisation program, followed by the highlights of the modelling language that has been implemented. We will address the modifications of the language after its introduction. We discuss which subset of the modelling language is implemented for the specific requirements of the SPYRO program and describe the functionality that has been added to the modelling language. We will give an example to illustrate the benefits of the added functionality. Finally we will conclude with the experience gained.

[1] SPYRO® Suite 7 is abbreviated to SPYRO in the rest of the paper.

1.1. Motivation and choices

The model equations, residuals, Jacobian matrix, sensitivity equations and derivatives in process simulators like SPYRO are hard coded in optimised source code. Generally speaking the computation speed of these equations is high and the flexibility to change these equations is low. Depending on the quality of the package, it offers the flexibility to model all particular configurations of the process, in this case steam pyrolysis. The drawback is obviously that other processes cannot be modelled and the available computation power cannot be exploited without coding of new model equations. The coding of new models is time consuming, error prone and specialised people are required. The addition of a symbolic modelling module expands the package to other model equations without the need of specialised people and where the modeller can focus on the model equations solely and not on the IT and mathematical issues. The above motivated us to integrate a Symbolic Modelling Definition (SMD) module into a dedicated equation-based flowsheet package for steam pyrolysis. This leads to the following requirements for the SMD module. The SMD should facilitate to define process models, described by a NLAE, DAE or PDAE system, without the need of detailed knowledge of IT and mathematical techniques. Typically a process that can be described with a small number of species (10–20), are continuously operated, can contain discontinuities, and apply all kinds of heat and mass transfer processes. The module should also have facilities to connect with existing legacy models.

The topics to consider prior to the implementation of such a module are:

1. Selection of a suitable language for the SMD module.
2. Integration structure of the SMD module with the existing flowsheet program.

The first topic is the selection of a language to define the model equations. The modelling language is the interface between the modeller and the computer-aided simulation tool. The third generation programming languages, such as C, C++, FORTRAN and Java, *language of computational procedures* [4], provide the maximum flexibility for the development of process models. These languages are tailored for the description of solution procedures for mathematical models, not necessarily for the models itself and require a significant amount of knowledge on programming, numerical mathematical techniques and time for design/coding and debugging of the models. Also the reusability of these models is inherently difficult as well as the modification to other applications of the programmed models [4]. Traditionally the *language of unit operations* has been applied by all modular flowsheet simulators. Despite the ability to supply user defined models this language of unit operations is restricted by the available models in the libraries and the dictated connection structure of the models. The equation-based packages provide the *language of mathematical equations* that overcomes the limitations of the procedural restrictions of the previously mentioned languages. This language allows the modeller to define the equations in a symbolic form, providing a focus on the model equations itself. The language embodies logical operators to facilitate conditional model definitions. Object-oriented programming concepts are implemented to organise and structure the mathematical description of the processes. The *language of process modelling*(also called *language of physicochemical phenomena*) has the ambitious goal to enable all chemical engineers to readily build and use models by supporting the model activity at the level of chemical engineering knowledge. Instead of defining the model equations, the modeller must define the assumptions taken, which species, phases, reactions, fluxes, flows are considered. He has to define the tasks, which describe the control actions and disturbances imposed on the processing system by its environment. In other words the interacting physicochemical phenomena are formulated. This language is still under development nevertheless it has a great potential since it would resolve some major deficiencies of current modelling tools [6].

The SPYRO program started as a package typically written in the *language of computational procedures*, evolved to a package using the *language of unit operations*, and subsequently to the *language of mathematical equations*. Obviously we consider both the language of equations and process modelling as the proper kind of language for the SMD module. Some of the available modelling tools are shown in Table 1 that features the language of mathematical equations and process modelling. These languages come in different forms, namely as a publication (PhD thesis, article, conference paper, open-source project, etc.), and as binaries (implemented in existing packages). All of the languages reported originate from academic research. A limited number have been commercialised. When a language is used that is published only, one needs to build the complete SMD module from scratch. This implies knowledge on programming, information technology, symbolic and discrete mathematics. The amount of work can be reduced if libraries are available, in the case of SPYRO several general purpose libraries for mathematical routines and all kind of utilities are present. The implementation of a language based on publications will generate the maximum flexibility since the module can be tailored to specific requirements, but will require a significant amount of effort. The use of an existing package (binaries) for the SMD module can be established through, amongst others, the CAPE-OPEN interface. This option will require a limited amount of effort but comes at a price of an additional license fee for the package. The flexibility will be limited because it depends on the willingness of the package supplier to add or change functionality to the SMD module. The application of an open-source SMD module, for example Modelica (http://www.modelica.org/), is conceptually flexible since the source code is available and can be established with limited effort. SPYRO is not open-source and therefore only an open-source project with a GNU L-GPL (GNU's Lesser General Public License) can be used for the SMD module, in all other cases special license should be obtained. We know from experience that a significant amount of effort is required for major changes of existing source code, especially when dealing with complex source code as required for the SMD.

Table 1
Selection of modelling tools (adapted from [19]).

Name of the system	Reference	Commercial package	Equation modelling language	Process modelling language
ASCEND	[1]		*	
ABACUSS	[30]	*	*	
DAESIM STUDIO	[13]	*		*
DYMOLA	[7]	*	*	
DYLAN	[18]		*	
gPROMS	[3,23]	*	*	
HPT	[38]			*
MODASS	[26]			*
MODEL.LA	[4]			*
MODELICA	[12]		*	
MODELLER	[37]	*		*
MODEX	[20]			*
MODDEV	[14]			
MODKIT	[5]			*
OMOLA	[21]		*	
PROFIT	[27]			*
PROMOT	[31,32]			*

In an industrial environment, reliability, flexibility, maintainability and cost of a new SMD module are important factors. The *process modelling languages* are still under development, and no need exists to research this kind of languages, and are therefore not (yet) applicable in our industrial setting. When the SMD module is created from scratch the cost of the module is proportional to the spent man-hours which are significant and difficult to estimate. The cost when connecting SPYRO to an existing SMD module is easily estimated because it resembles roughly the license cost and the cost to create the interface. On the other hand the licence fees for a reliable, tool are significant and the flexibility to change or add new functionality is dependent on supplier. This led to the decision to create a SMD module from scratch because it gives the best reliability, flexibility and maintainability. With hindsight we observed that the costs to build the SMD module were modest, approximately a half man-year.

The available equation modelling languages do not differ much conceptually, although the syntax is quite different. Also the languages are all accessible through publications and are considered equally difficult to implement. We observed the application of the individual languages both in academia and industry to distinguish between the available languages. The gPROMS language, as proposed by [3], is selected because it has proven its value both in industry and academia. The language is also closely related to ABACUSS [11] and the syntax is related to SPEEDUP [24], both are applied in industry and academia as well.

The second topic is related to the way the SMD module is integrated in the existing equation-based flowsheet package. In an equation-based program the solver and model equations are strictly separated. The communication between solver and models is basically done by an unknown vector, residual vector and derivative matrix (Jacobian). The SMD module will also generate unknowns, residuals and Jacobian matrices. The hard coded model equations and SMD equations can be solved with or without usage of each others variables. When the variables of both modules are integrated, the syntax needs to be created to use variables of the hard coded models in equations defined by the SMD module and vice versa. If they are not integrated the solver can determine the solution of the hard coded models and SMD module at the same time or separately. Our main objective is to model processes other than steam pyrolysis therefore this integration of variables is deferred to a latter stage. The current model library is considered rich enough for the steam pyrolysis process. As a consequence we either offer models, defined in the SMD module, or hard coded models, to the solver. A second aspect of the second topic is the mode of operation, the SMD module can be build as an intermediate code generator, such as SPEEDUP [24] and DynoPC [17], or as an interpreter, such as gPROMS [23], and ASCEND [1]. The first operation mode requires code generation (in e.g. C/C++, or FORTRAN), compilation, linking which is time consuming even for a relative small number of equations. In the second mode the SMD is interpreted into data structures which can be evaluated directly without intervention of other programs. The advantage of this mode lies in the elimination of the compilation and linking steps, but has the disadvantage of being more memory intensive and less computationally effective than object code. Nowadays computer power make is possible to apply the interpreted mode of operation. When in the future models are built that cause performance issues in the SMD module we can develop a utility to generate source code, compile and link this as in the first operation mode.

2. Description of the SPYRO® Suite 7

The program features the simulation/optimisation of a flow sheet with predefined models, typical for steam pyrolysis: Firebox, Radiant Coils, Transfer-line exchangers, etc., and general purpose models such as: Feed, Product, Splitter, etc. The program is divided into several functional sections, as schematically shown in Fig. 1. The Data Manager is the central section, the "spider in the web", controlling the data flow/storage of the program. The Graphical User Interface or Visualiser takes care of the visualisation of both the input and the output, this data is stored in an XML-file. The Solver-Optimiser Manager

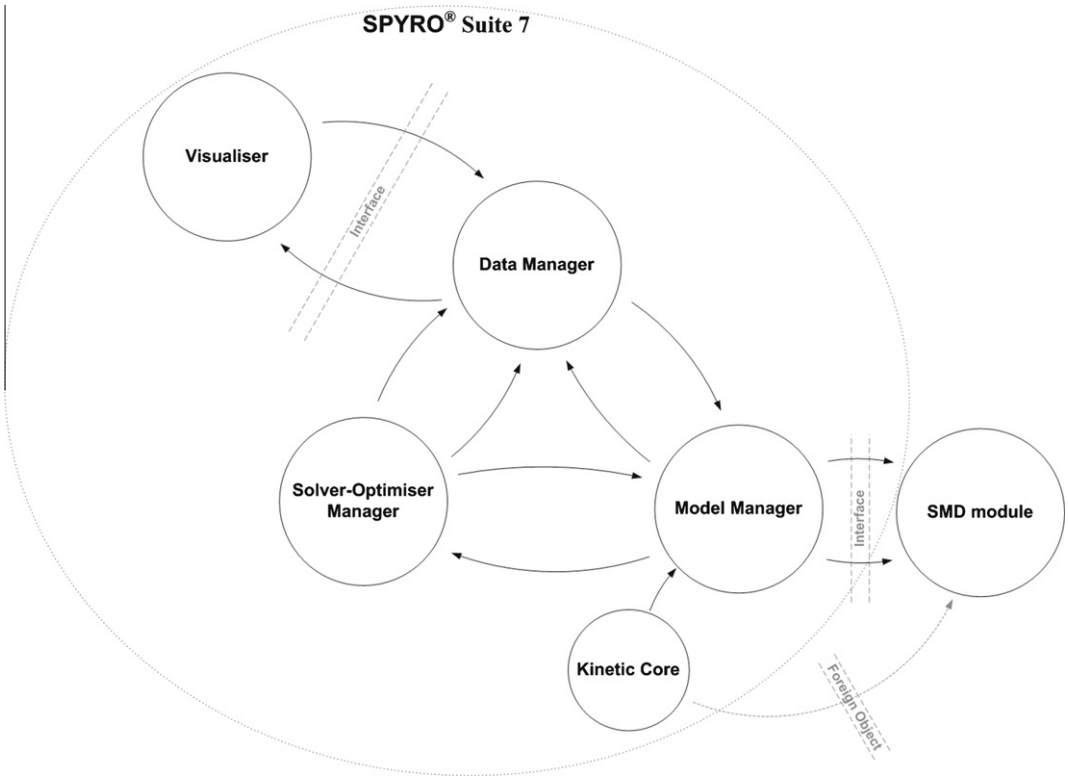

Fig. 1. Functional decomposition of the SPYRO program.

contains all the available numerical mathematical procedures for solving both linear and non-linear systems of equations and optimisation problems.

The Model Manager contains controls all the predefined models, residuals, analytical derivatives, initialisation output generation, etc.

The core of the chemical know-how of the SPYRO program are the equations for the pyrolysis and its kinetics (see "Kinetic Core" in Fig. 1). These are hard coded. The steam pyrolysis is described with a free radical mechanism, which is inherently stiff due to the different reaction velocities of the radicals and the molecular components. The latest kinetics scheme contains approximately 7000 reactions between 210 molecular components and 40 radical components. The steam pyrolysis process is described by a set of partial differential algebraic equations. The temporal dependency describes only slow phenomena, the formation of coke inside the radiant tubes (typical time scale is 40–60 days) and the number of temporal dependencies is small compared to the spatial dependencies. The numerical challenges are located in the description of these spatial properties, originating from the stiffness of chemistry model. The spatial dependencies are approximated effectively with the orthogonal collocation on finite elements technique [17]. Apart from the stiffness and other non-linear behaviour, the largest computation efforts is for the direct linear solver to perform a decomposition in reasonable time ($\approx$10 s) on a regular desktop PC (P4, 3.4 GHz, 2 GB RAM). The large Jacobian matrices (120,000 $\times$ 120,000 with 4×10^6 non-zeros) originate from the large number ($\approx$250) of equations of continuity and the presence of a 3D-Firebox model. The used solution system is such that we can solve 200,000 equations. This is sufficient for the pyrolysis calculations and also for the added SMD module.

2.1. Modelling error reduction

Several functionalities have been implemented to minimise the errors made by the modeller. This reduces the debugging time of the models submitted to the solving process. The functionalities vary from basic checking to the application of mathematical techniques in the core of the solver. Our vision is to provide clearly the relevant information to the modeller, both textually as well as graphically, based on mathematical progress indicators.

The first layer for checking is the visualiser (Fig. 1) that should be designed such that information is intuitively grouped and ordered logically, for example geometrical data should be separated from operational data (inlet temperature, flow rate, etc.). The Data Manager features a three level check procedure. The first level features the checks performed on one entity in a model, for example the lower, upper bound, the value of the sum of an array (mol fractions), etc. On the second level the

consistency of a full model is checked, for example the specification of physical property systems with different reference states within one model. On the third level the consistency between different models is checked, for example the sequence of the sub-models of a radiant coil which have a predefined sequence. During the initialisation of the Model Manager, prior to the solving process, checks are performed whether the system is well-posed. Notification messages are issued, with possible root causes of these issues. When the modeller has defined a case that successfully passed all these checks and is submitted to the solution procedure, errors in the models can still be present.

The first type of modelling errors is of a numerical nature. The permutations applied in the direct linear solver, not only facilitate efficient factorisation but also give insight in the well-posedness of the problem. The transversal permutation yields information on the *structural singularity* of the system [29]. The detection whether a proposed system of equations is *numerical structurally singular* is found easily in the pivoting process of the linear solver.

The second type of errors is an inconsistent (unphysical) definition of the problem, for example: description of a compressible flow through a tube where the specified flow rate is too large for the tubes geometry (pressure drop getting too high). The root cause is usually found by the inspection of different sets of output generated with a different number of iterations. The amount of data that needs to be inspected can be large and therefore the discovery of the root cause of the problem can be time consuming. Simulation programs report usually mathematical indicators to give insight in the progress of the solution procedure, such as damping factors, norm of residuals and correctors. This type of information does not give any insight in specific variables and residuals which possibly give rise to poor convergence behaviour as described above. In SPYRO we have added features to reduce the time to discover these errors of the second type. This is accomplished by reporting specific variables and residuals with the largest change during the solution procedure, besides the usual indicators. The following items benefit the modeller in the diagnostic phase (see also Fig. 2):

1. The variable with the largest violation of its bounds (used to compute the maximum damping factor).
2. The residual with the largest absolute value.
3. The variable with the largest absolute corrector step.
4. The variable with the largest relative corrector step (relative to the value of the variable).

Although this additional information gives valuable information, graphical intermediate results provide even faster and more insight in the solution procedure. Monitoring several intermediate results enables the modeller to draw conclusion on particular root causes of poor convergence behaviour of the model, for an example of the looking view see Fig. 2.

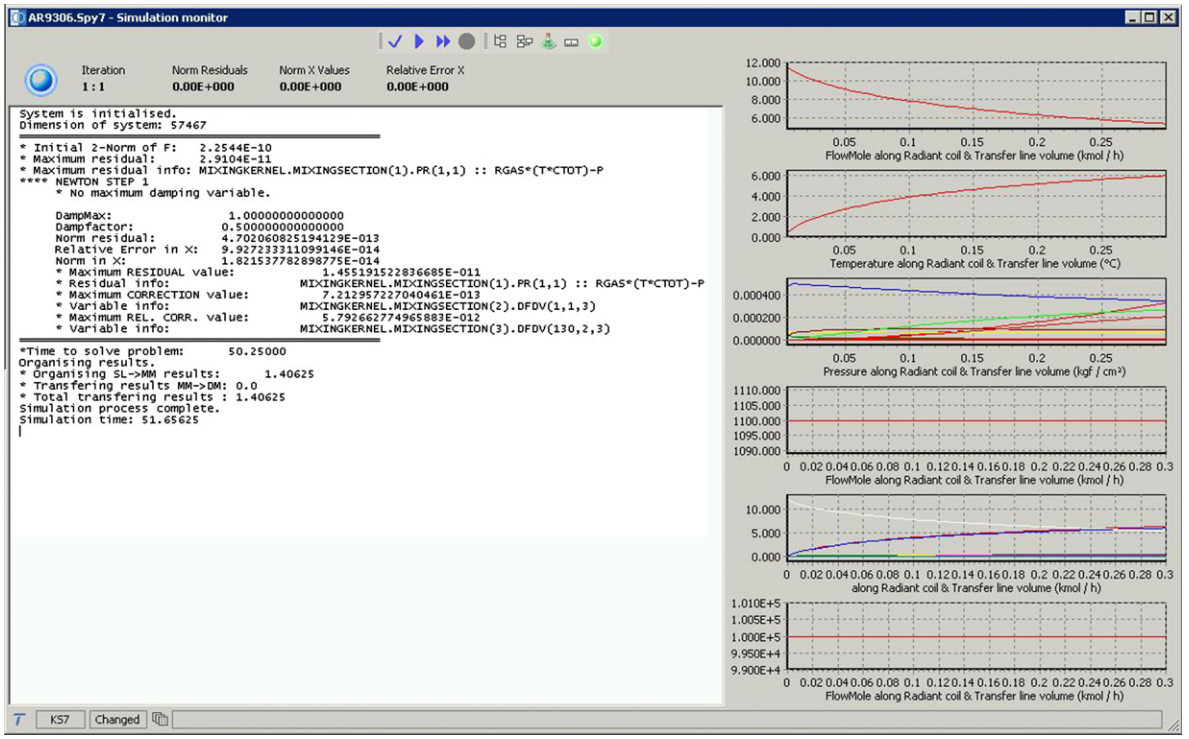

Fig. 2. Example of combined numerical solver output and graphical intermediate model results.

3. Description of the modelling language

We have selected the language as defined by Barton [3], called gPROMS. The commercial implementations of this language are known under the names gPROMS® (distributed by Process Systems Enterprise (PSE) Limited) and ABACUSS II (continued under the name JACOBIAN and distributed by Numerica Technology). For convenience we will label the language used in these programs as defined by [3]: gPROMS.[2] This language is designed to formulate dynamic simulations of systems with combined discrete and continues processes of arbitrary complexity. The language proposed in its original form consists of four distinct entities, namely the *DECLARE*, *MODEL*, *TASK* and *SIMULATION* entity. The basic variable types, in terms of lower and upper bound and default value and stream types are defined in the *DECLARE* entity and are globally available to all model entities. *MODEL* entities describe both the continuous physicochemical mechanisms governing the time dependent behaviour of unit operations and any discrete changes resulting from these mechanisms. The *TASK* entities describe the external control actions or disturbances imposed on the modelled system. A *SIMULATION* entity defines a full dynamic simulation, where tasks are applied to instances of model entities [3]. The definition of the language in a Backus–Naur From can be found in appendix B of the PhD thesis of [3].

A model for a complex process is constructed efficiently by the introduction of structure through the definition of several interconnected models. gPROMS provides three main functionalities in order to provide this structure.

1. The basic building block is a so-called primitive or root model, which contains no sub-models. Primitive models are constructed by the definition of attributes such as: parameters, variables, streams and equations [3].
2. The second functionality is hierarchical sub-model decomposition [10]. A model may be declared as a collection of interconnected sub-models. Typically the connections represent physical streams of fluxes of mass,energy, electricity, etc.
3. The third functionality that is offered by gPROMS is the hierarchical model development through inheritance. The inheritance concept is well known from object-oriented programming languages. An inheritance hierarchy can be represented schematically by a directed tree, where the vertices signify the models and the edges signify the relationship ‘‘inherits from’’.

Languages are subject to changes and also gPROMS has changed after its introduction in 1992. We will highlight the changes relevant for our implementation. Developments in the (dynamic) optimisation community led to the need to have an entity for the definition of optimisation problems. The language is developed by two separate groups, gPROMS® [25] and ABACUSS [8], both groups added an *OPTIMISATION* entity as a fifth main entity. This is required to define typical relations for optimisation problems, for example: objection function, (in)equality constraints and control variables. In the gPROMS® group the emphasis is on adding/changing a *SIMULATION* entity to an *OPTIMISATION* entity and in the ABACUSS group the *OPTIMISATION* entity is equal to a *SIMULATION* entity with additional features to define the typical optimisation features. The main *SIMULATION OPTIMISATION* entities show a differentiation between the two groups, through the naming of the entities and the attributes available inside the entity.

The language used by equation-based packages overcomes the limitations of the procedural languages. Nevertheless procedural functionality is still required, for example, to incorporate existing legacy code such as physical property packages. The way how the language enables to connect with external procedures stored in libraries is another explicit difference. In ABACUSS a formal robust, programming approach is taken, to minimise errors, by the definition of the *EXTERNAL* entity that defines the connection with external procedures [30]. In gPROMS® the ‘‘foreign objects’’ have been introduces within the *MODEL* entity, which is less formal and more flexible. A parameter defines which external library should be accessed and procedures within this library are accessed such as variables in other models. If, for example, the parameter *P* defines the FOREIGN_OBJECT ‘‘ThermoPack’’ then the procedure to compute the liquid enthalpy is addressed within the model as follows: *P.LiquidEnthalpy*(*T*, *p*, *x*).

4. Language used

In this paragraph we address which elements of the gPROMS language will be used to fulfil the symbolic modelling requirements within SPYRO. Additionally, the new attributes will be discussed for the main entities: *DECLARE*, *MODEL*, *TASK*, *SIMULATION* and *OPTIMISATION*.

The *TYPE* attribute from the *DECLARE* entity, is implemented with an additional feature to have besides the upper and lower bound also a soft lower and soft upper bound, that are required by the numerical procedures within SPYRO. When these soft bounds are omitted they are assumed to be equal to the hard bounds. The *STREAM* attribute of the *DECLARE* entity is not implemented, since it is not an essential functionality.

The attributes, *PARAMETER*, *UNIT*, *VARIABLE*, *SET*, and *EQUATION* of the *MODEL* entity have been applied. A minor change to the original gPROMS is the option to have a *PRESET* attribute in a *MODEL* entity, to make the initialisation of the variables more convenient. At this moment there is no requirement to model discontinuously operated processes and therefore we did not implement the CASE-equations and the IF-equations (also known as conditional equations), in the *MODEL* entity.

[2] When we use the registered trademark token, ®, we refer to the program distributed by PSE Limited.

The foreign objects can be used to define discontinues functions instead of the IF- and CASE-equations. This implies as well that temporal derivatives are not yet supported. Nevertheless DAE and PDAE systems are solved and optimised in an equation-based fashion, as for example done by Biegler and his co-workers [17]. For this purpose several additional functions, besides the standard functions such as: *sin*, *cos*, *erf*, *min*, *max*, *sigma*, etc., for Orthogonal Collocation on Finite Elements (OCFE) to compute the discretisation matrix, *ocmat*, location of the collocation points in a finite elements, *ocrloc*, and the distribution of the finite elements, *felr*.

The *TASK* entity is not implemented because there is no need to model discontinuously operated processes. The *SIMULATION* entity features all the attributes of the *MODEL* entity plus additional attributes for the output (*MONITOR*) and assignment (*ASSIGN*) of variables. The typical attributes for the time integration have been omitted such as, *SCHEDULE, INITIAL*. We adopted the ABACUSS approach for the *OPTIMISATION* entity because it is more in line with the whole language setup as defined by [3]. The *OPTIMISATION* entity is equal to a *SIMULATION* entity with additional attributes for the specific optimisation requirements, such as *MAXIMISE*, *MINIMISE*, *INEQUALITY*, *TIME INVARIANT*, etc.

The features to improve the modelling efficiency in SPYRO proved fruitful. Therefore these items had to be incorporated into the language. We have added to the *SIMULATION* and *OPTIMISATION* entity two new attributes to enhance model debugging and simulation and optimisation efficiency. The first attribute is the display of intermediate model results to inspect the progress of the simulation and optimisation. As mentioned SPYRO features several functions to debug models. For this purpose we allow the modeller to specify 6 intermediate graph attributes. Fig. 2 shows, besides the usual numerical solver output the graphical intermediate model results. Within the *GRAPH* attribute the results of a series of consecutive expressions are interpreted as data sets for the graph where the first value of the result of an expression is considered to be the *x*-value, and the other values as *y*-values. Such a series of consecutive expressions is ideally defined with a FOR-equation, for example shown by Fig. 3. Note the subtle difference in the END-statement where we require ENDWITHIN to close a WITHIN statement, this is also required for the END statement of the *MODEL*, FOR-equation, etc. Reduction of errors is important in an industrial setting and for this reason we implemented this feature.

The second new attribute that is added, *SCENARIO*, defines consecutive simulations and optimisations to inspect the sensitivity of the results to model parameter values. This sensitivity is computed by performing a series of simulations or optimisations with different values for the parameters of interest. An example on how to define 2000 simulations within a *SCENARIO* attribute is given by Fig. 4.

The output of the sensitivity is augmented with the values of the parameters in the *VARY* block and a convergence indicator of the solver or optimiser.

```
GRAPH1
  WITHIN ModelOne DO
    FOR i:=1 TO nSections DO
      Time(i),Temperature(i);
    ENDDO
  ENDWITHIN
```

Fig. 3. Example how to define a graph to display intermediate results during simulation/optimisation.

```
SCENARIO
  VARY
    # Parameter:= First value, last value, number of points;
    Length  :=  1,  42, 200;
    Diameter:=0.1, 0.5,  10;
  OUTPUT
    Yield, Conversion;
```

Fig. 4. Example of a *SCENARIO* attribute to define 2000 simulation to inspect the dependency of the yield and conversion to different geometry values.

```
PARAMETER
  ST    AS ARRAY(2)  OF  SPLINE
SET
  ST:=SPLINECONSTR([0,1],[[0, 84],[1,2]],'cubic','linear');
EQUATION
  42 = SPLINEVALUE(Solution,ST(1));
```

Fig. 5. Example of a *SPLINE* attribute to define an array of two cubic splines with linear extrapolation, the construction and application of the spline in an equation.

Incorporation of external procedures are defined with the "foreign objects" concept from gPROMS® since this yields the most flexible options for the users while the responsibility to deliver robust external libraries is delegated. Robust external procedure functionality is provided by splines that have been incorporated in the gPROMS language. We have added the *SPLINE* attribute (see Fig. 5), additional to the *INTEGER*, *REAL*, *LOGICAL* and*FOREIGN_OBJECT* attributes of the *PARAMETER*section. The functions SPLINECONSTR and SPLINEVALUE have been added to respectively construct and evaluate a spline, the input and output operate both on scalars as on arrays like all other expressions.

5. Implementation of symbolic modelling

Fig. 1 shows the functional sections of the program, as shown, the SMD module is implemented as a separate module. The Model Manager provides the residuals and Jacobian matrices, when an instantiated SMD case is simulated/optimised the residuals and the Jacobian matrices are supplied by this extension. Currently no features are present to combine the predefined models with the SMD models. The kinetic core of SPYRO can be used through the "foreign object" functionality of the SMD. This has been tested on the model as explained in the second example with a foreign object for the production rate consisting of 3000 reactions between 150 species.

The SMD module is divided into several functional parts, as shown by Fig. 6. During the initialisation of the Model Manager the SMD input file is parsed and instantiated. The parser translates the input file into data structures that contain the *MODEL*, *SIMULATION* and *OPTIMISATION* entities (see Fig. 6), and it reports in case of errors, what and where the input file items need to be corrected. To facilitate parsing of the input file within an users preferred text editor, we added a separate

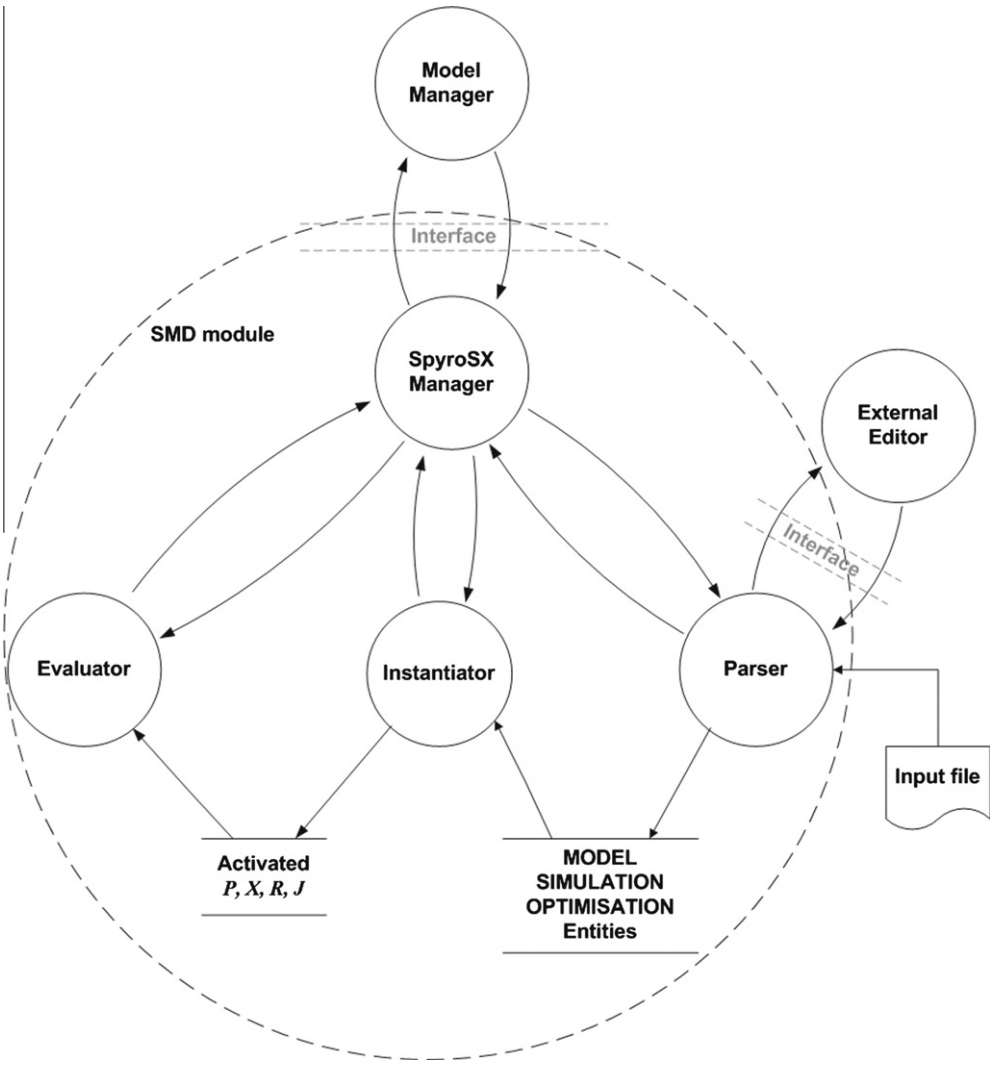

Fig. 6. Functional decomposition of SMD module, see Fig. 1 for the connection with SPYRO (SMD module).

interface to check the structural, semantic or lexical errors within the input file. When the parsing is successful the generated data structures are instantiated, and an active parameter array (P), an active variable array (X) and active equation/partial derivative arrays (R/J) are created (see Fig. 6). Additionally, the instantiator performs semantic checks which have been deferred by the parser due to the extensive model parameterisation applied in gPROMS. For example, the size of an array can be defined by parameters and during the process of instantiation the ultimate values of the parameters are known. Checks to determine, amongst others, dimensional correctness need therefore to be postponed until instantiation of the models. Appendix A denotes some details on how symbolically defined equations are translated into data structures that can be evaluated to yield numerical values and partial derivatives.

6. Illustrative example

We present a simple and a complex example to illustrate the gained flexibility and the exploitation of the computation power of SPYRO for solving other models than the predefined flowsheet models. Although we have deferred to a latter stage, the integration of the predefined variables with SMD variables, the first example demonstrates that we can combine these two via a *FOREIGN_OBJECT*. During the implementation of the second example we added the functions to apply the OCFE technique more easily. This shows the flexibility when one develops such tools in-house. This is particular important in an industrial setting where efficiently is more important.

6.1. Temperature profile comparison

In this example we will illuminate the use of the *SPLINE* and *FOREIGN_OBJECT* attributes. We are interested in the difference of ethylene yield for isothermal and some measured temperature profile for the steam pyrolysis of ethane with an outlet temperature of 850 °C. The equation of continuity for the kth component is described by

$$\frac{dF_k}{dV} = R_k(F,T,P) \qquad V = 0 \qquad F = F_0 \tag{1}$$

The chemical know-how of SPYRO is in its kinetic models for the production rates, R, of the pyrolysis process (see Fig. 1). These procedures have been made accessible for the SMD module through the application of a foreign object. Fig. 7 shows the declaration of foreign object, FO, which is similar to other parameters. In the *SET* section the FO is assigned to the desired external library, "KS7FOB.DLL". The spline parameter, ST, for the measured temperature profile is declared similar to other parameters and constructed in the *SET* section. In the equation section the temperature at a collocation point, iCol, in finite element, iFE, is defined equal to a value of the spline function, SPLINEVALUE. Fig. 7 shows as well the definition of the production rates, R, equal to external procedure "ProdRates" where the differential Eq. (1) is descretised with OCFE. Fig. 8 shows the results of the ethylene versus ethane conversion for the isothermal and the measured temperature profile.

```
PARAMETER
  RelZ AS ARRAY(42) OF REAL
  ST   AS              SPLINE
  FO   AS              FOREIGN_OBJECT
SET
  # Define relative coordinate
  RelZ:=[0, 0.3, ..., 1];
  # Define corresponding temperature difference
  RelTemp:=[0, 0.42, ..., 1];
  # Construct the temperature spline
  ST:= SPLINECONSTR(RelZ,RelTemp,'qubic','linear');
  # Set the foreign object to the KS7FOB.DLL
  FO:="KS7Fob"; # The foreign object for ProdRates
EQUATION
  T(iCol,iFE) = Tlow+ (Tval-Tlow)*
              SPLINEVALUE(Volume(iCol,iFE)/VolTot,ST);
  FOR iComp:= 1 TO noComp DO
    SIGMA(OCFEmat(iCol,:)*F(iComp,:,iFE))=VolTot*
        FO.ProdRates(F(:,iCol,iFE),P(iCol,iFE),T(iCol,iFE));
  ENDDO
```

Fig. 7. Example on the use of *SPLINE* and *FOREIGN_OBJECT* attribute in a SMD input file for the comparison of ethylene yields for a isothermal and measured temperature profile.

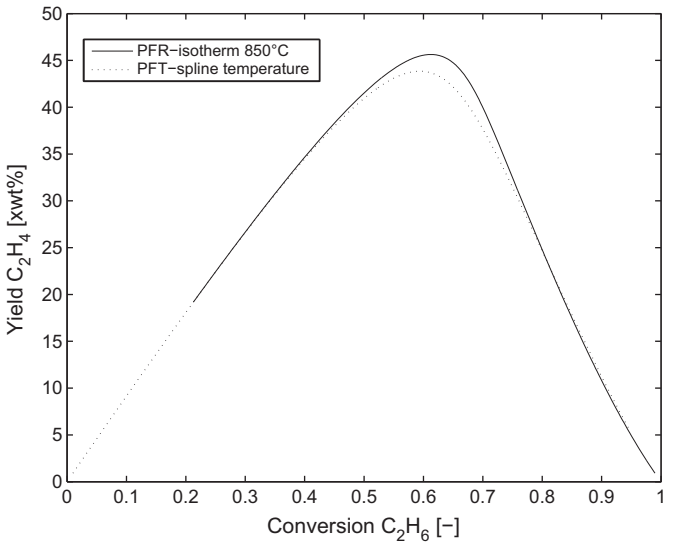

Fig. 8. The conversion of ethane versus ethylene yield for a outlet temperature of 850 °C.

```
SET
  # Discretisation matrix
  OCFEmat  :=OCMAT(nCol,Alpha,Beta);
  # Finite element relative length's
  # When Parm <0 Left side has higher FE density
  RelLenFE :=FELR(nFE,Parm);
  # Relative location collocation points within a FE
  RelLocCol:=OCRLOC(nCol,Alpha,Beta);
EQUATION
  FOR iComp:= 1 TO noComp DO
    SIGMA(OCFEmat(iCol,:)*F(iComp,:,iFE))=RelLenFE(iFE)*VolTot*
     (Prodrate(iComp)+MixIn(iCol,iFE,iComp-MixOut(iCol,iFE,iComp));
  ENDDO
SCENARIO
  VARY
    CaExit  :=  0.1,  0.99, 300;
  OUTPUT
    Cout,Qv;
```

Fig. 9. Example of a *SCENARIO* attribute and OCFE functions to define 300 optimisations for the determination of an attainable region.

6.2. Distributed reaction and mixing

In this example we demonstrate the effectiveness of the *SCENARIO* attribute in the *OPTIMISATION* entity and the available functions for OCFE for a more mature problem. We use the distributed reaction mixing (d-RMix) model [34], Eq. (2) to determine the attainable region [15] for the [33] reaction scheme.

$$\frac{\partial F_{k,V}}{\partial V} = R_{k,V} + L(V)F_0, k - K(V)F_{k,V} + \int_0^{V_t} (F_{k,v}M(v,V) - F_{k,V}M(V,v))dv \tag{2}$$

The first term of Eq. (2) represents the convective transport. The second term covers the net rate of formation of the kth component. The third term indicates the distributive injection of the feed over the total available reaction volume. The fourth term accounts for a distributive removal of reaction material along the reaction coordinate. The fifth and the sixth term model the distributive re-allocation of reaction material from one location (V) to another one (v), this is called "distributive mixing". The reaction scheme of the [33] is given by (3). Additional information on the model formulation and the solution procedure can be found in Appendix B.

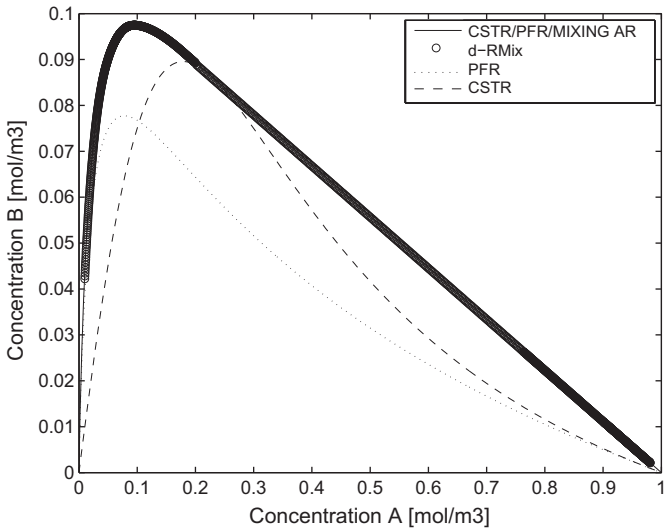

Fig. 10. Attainable region for [33] system [1, 1, 10] by CSTR/PFR/MIXING and *d*-RMix.

$$\begin{aligned} &A \xrightarrow{k_1=1} B \xrightarrow{k_2=1} C \\ &2A \xrightarrow{k_3=10} D \end{aligned} \tag{3}$$

Fig. 9 shows the definition of the discretisation matrix, OCFEmat, with the function OCMAT, which is parameterised with the number of collocation point, nCol, and the parameters for the Jacobi polynomials. To influence the density of collocation points we want to compute a series of finite element length which are smaller on the left hand side or on the right hand side of the domain. The function FELR enables this with one assignment in the *SET* section. For similar reasons the relative location of the collocation points within a finite element can be retrieved with the OCRLOC function.

Fig. 10 shows the attainable region determined with the geometric technique and the *d*-RMix (The attainable region is the region enclosed by the boundary and the *x*-axis in Fig. 10). The objective of this optimisation was to maximise the concentration of *B* at the exit of the reactor by adjusting the mixing kernel *M* and feed distribution function *L*. The effluent removal function, *K*, was defined zero, the total volume was set to an arbitrary value of 0.3 m^3, and the exit concentration of *A* was fixed to the desired value. This is possible by using the volumetric flow rate entering (constant density) the reaction volume as a degree of freedom. Each point in the profile for the *d*-RMix is the outcome of an optimisation problem, the large number of points demonstrates the effectiveness of the *SCENARIO* attribute as shown in Fig. 9.

7. Conclusions

Simulation packages evolve with the scientific progress made in the area of the modelling languages. This can be observed on the steam pyrolysis simulation standard, SPYRO that started as a simulation tool for the process side of the radiant coils solely, progressed to a sequential modular flow sheet package and has evolved further to an equation-based flow sheet package for this process.

This work denotes the experience gained on how the equation-based solving power of the program is opened up to any model through the integration of a Symbolic Model Definition (SMD) module. The selection of a language for the SMD module in an industrial setting is somewhat conservative since it should be reliable, flexible and maintainable at reasonable cost. The selection of the process modelling languages that are still under development (amongst others MODEL.LA, MODKIT) is not opportune because of the uncertainties accompanied by being under development. The symbolic module yielded additional flexibility for SPYRO. The main engine retained its computational efficiency, but it is now available to different sets of problems. It has effectively reduced the time to develop alternative models for a variety of units in the process industry, where SPYRO is used until now only for the reaction kinetics.

The descriptive power of the available equation-based modelling languages does not differ significantly. The implementation of the available languages is considered equally difficult. The gPROMS language [3] is selected because is has proven itself over the years, both in industry as in academia.

The implementation was relative straight forward and took on the order of a half man-year. The language has been adapted to the specific requirements of the program, that is, supply the modeller with more information during the solution procedure. This means not only the numerical indicators of the mathematical progress but also indicators, both textual as graphical (*GRAPH* attribute), useful to the modeller to signify possible errors inside the model. The numerical algorithms allow soft lower and soft upper variable bounds, these have been added in the DECLARE entity to the *TYPE* attribute.

Series of simulations/optimisations are straightforwardly defined with the *SCENARIO* attributed.

The *FOREIGN_OBJECT* attribute provides a flexible connection to external executable libraries while the responsibility to create robust libraries is delegated. This attribute requires programming knowledge and effort in e.g. C/C++ or FORTRAN 95. A robust way to incorporation of external procedural functionality is provided by the *SPLINE* attribute which is a natural extension of the language.

The symbolic module yielded the required additional flexibility for SPYRO, it provided a larger appeal on the underlying software as well through which its quality improved.

In the future we will implement dynamic optimisation based on an integration based method (implementation of the temporal derivative defined in gPROMS, $), such as BDF. This will catalyse the implementation of discrete event modelling, for example decoking of transfer-line exchangers. The addressing of the variables defined by the hard-coded models in the SMD module and vice versa is also on the to-do list. We will add the *CONNECTION or STREAM* attribute to the *DECLARE* entity in order to more easily manipulate the passing of variables between modules. The functional extension of SPYRO is reached and therefore no graphical interface for the SMD module is foreseen.

Acknowledgements

This PhD project is supported by Technip Benelux B.V. The fruitful discussions with Johan Grievink, Frank Ouwerkerk and Simon Barendregt are appreciated.

Appendix A. Symbolic mathematics

The core of the SMD module is the procedure to evaluate symbolic algebraic expressions. Our implementation, in FORTRAN 95, is based on a binary-tree representation of algebraic equations (see e.g. [22,28]). The tree of the general algebraic expression (A.1) is shown Fig. A.11.

$$r = f(x)^{g(x)} \tag{A.1}$$

The vertices in the tree represent operators, such as addition, subtractions, multiplication, functions, etc., and the edges denote the sequence of evaluation, often referred as LHS and RHS (Left/Right Hand Side) of the operator. The root vertex represents the result of the algebraic expression and the leaves of the tree represent the individual variables, parameters, constants, etc., in between the vertices represent the operators.

$$\frac{\partial r}{\partial x} = f(x)^{g(x)}\left(\frac{\partial g}{\partial x}\ln(f(x)) + \frac{g(x)}{f(x)}\frac{\partial f}{\partial x}\right) \tag{A.2}$$

Once the expression tree of an algebraic expression is available the partial derivative expression tree can be constructed by recursively applying the rules of differentiation, for example Eq. (A.2) which is the differentiation rule of Eq. (A.1). During the construction of the derivative tree of (A.1), Eq. (A.2) needs to be added to the derivative tree, the vertices that are added are shown in Fig. A.12. The algorithm to construct a derivative tree from a residual tree is given by Fig. A.13.

The evaluation of the expression trees is performed from the leafs to the roots of the tree. This is accomplished with a depth first evaluation procedure, as shown by Fig. A.14.

Appendix B. Distributed reaction mixing model

Molar-balance for kth component, along reactor coordinate V:

$$\frac{\partial F_k(V)}{\partial V} = R_k(C,T,P) + L_j(V)F0_k - K(V)F_k(V) + \int_0^{V_t}(F_k(v)M(v,V) - F_k(V)M(V,v))dv. \tag{B.1}$$

All *HC* have the same feed distribution function $L_j(V)$ while steam has an independent one, j = *HC* for $k \neq H_2O$ and $j = H_2O$ for $k = H_2O$. Boundary condition for Eq. (B.1) are given below:

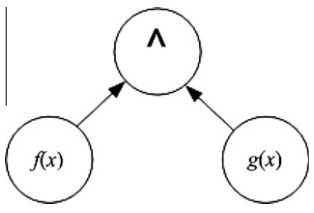

Fig. A.11. Example of residual represented by a binary tree.

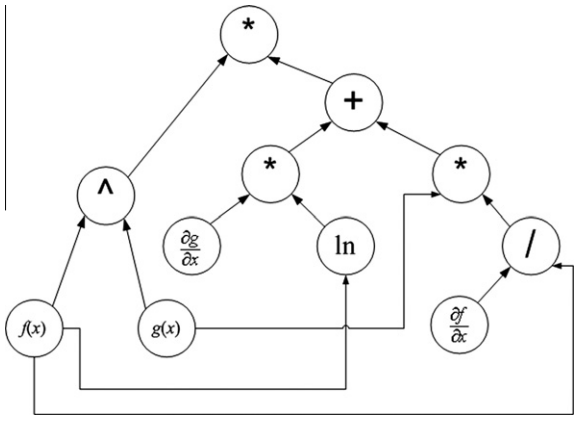

Fig. A.12. Binary tree of differentiation rule for Eq. (A.1), or its tree in Fig. A.11.

```
DIFF( R(Vertex), D(Vertex) )
  SELECT CASE(R(Vertex).Operator)
  CASE(CONSTANT)
    RETURN D(Vertex).Value=0
  CASE(VARIABLE)
    RETURN D(Vertex).Value=1
  CASE(PLUS)
    ADD D(Vertex).LHS
    ADD D(Vertex).RHS
    D(Vertex).Operator=PLUS
    DIFF( R(Vertex).LHS, D(Vertex).LHS )
    DIFF( R(Vertex).RHS, D(Vertex).RHS )
    RETURN
  CASE( )
        :
    Similar code for other operators
        :
  ENDCASE
END DIFF
```

Fig. A.13. Recursive algorithm for the creation of a derivative tree, *D*, from a residual tree, *R*.

$$F_k(V) = \phi_m \frac{x_{wt,k}}{Mw_k} \quad \text{for } V = 0. \tag{B.2}$$

Feeds weight fraction composition:

$$x_{wt,k} = \begin{cases} \frac{SDR}{1+SDR} & k = H_2O, \\ \frac{x_{wt,HC,k}}{1+SDR} & \text{else}, \end{cases} \tag{B.3}$$

where

$$\sum_{\substack{k=1 \\ k \neq H_2O}}^{n} x_{wt,HC,k} = 1. \tag{B.4}$$

```
Real EVALUATE(T(Vertex) )
  SELECT CASE(T(Vertex).Operator)
  CASE(CONSTANT)
    RETURN T(Vertex).Value = 42
  CASE(VARIABLE)
    rval = GET_FROM_X(T(Vertex).Position)
    RETURN T(Vertex).Value = rval
  CASE(PLUS)
    lhs_val = EVALUATE( T(Vertex) LHS )
    rhs_val = EVALUATE( T(Vertex) LHS )
    RETURN lhs_val + rhs_val
  CASE( )
        :
    Similar code for other operators
        :
  ENDCASE
END EVALUATE
```

Fig. A.14. Evaluation algorithm of a tree, *T*.

Change of residence time with the reaction volume coordinate (V):

$$\frac{d\tau}{dV} = \frac{\rho(V)}{\sum_{m=1}^{n} F_m(V) Mw_m}, \quad V = 0, \quad \tau = 0, \tag{B.5}$$

where the density is defined by the ideal gas law. The relation between the molar flow rate and the concentration is defined by the ideal gas law as well:

$$C_k = \frac{F_k}{\sum_{m=1}^{n} F_m} \frac{P}{R_g T} \tag{B.6}$$

and

$$\rho = \sum_{m=1}^{n} c_m Mw_m. \tag{B.7}$$

References

[1] K. Abbott, Very large scale modelling, PhD thesis, Carnegie-Mellon University, Pittsburg, Pennsylvania, 1996.
[2] S. Barendregt, P. Valkenburg, E. Wagner, M. Dente, E. Ranzi, History and recent developments in SPYRO, a review, In Pre-print Archive – American Institute of Chemical Engineers [Spring National Meeting], New Orleans, LA, United States, 2002, pp. 2497–2537.
[3] P. Barton, The modelling and simulation of combined discrete/continuous processes, PhD thesis, Imperial College of Science, London, 1992.
[4] J. Bieszczad, A framework for the language and logic of computer-aided phenomena-based process modelling, PhD thesis, Massachusetts Institute of Technology, 2000.
[5] R. Bogusch, B. Lohmann, W. Marquardt, Computer-aided process modelling with MODKIT, Computers and Chemical Engineering 25 (2001) 963–995.
[6] I. Cameron, G. Ingram, A survey of industrial process modelling across the product and process lifecycle, Computers and Chemical Engineering 32 (2008) 420–438.
[7] F. Cellier, H. Elmqvist, Automated formula manipulation supports object-oriented continuous-system modelling, IEEE Control Systems 13 (2) (1993) 28–38.
[8] J. Clabaugh, The ABACUSS II syntax manual. Technical report, Massachusetts Institute of Technology 2002.
[9] M. Dente, E. Ranzi, A. Goossens, Detailed prediction of olefin yields from hydrocarbon pyrolysis through a fundamental simulation model (SPYRO), Computers and Chemical Engineering 3 (1979) 61–75.
[10] H. Elmquist, A structured model language for large continues systems, PhD thesis, Lund Institute of Technology, 1978.
[11] W. Feehery, P. Barton, Dynamic simulation and optimisation with inequality path constraints, Computers and Chemical Engineering Supplement 20 (1996) S707–S712.
[12] P. Fritzson, Modelica – a language for equation-based physical modeling and high performance simulation, Applied Parallel Computing 1541 (1998) 149–160.
[13] K. Hangos, I. Cameron, Process Modelling and Model Analysis, Academic Press, 2001.
[14] A. Jensen, R. Gani, A computer aided modeling system, Computers and Chemical Engineering Supplement 23 (1999) S673–S678.

[15] S. Kauchali, W. Rooney, L. Biegler, D. Glasser, D. Hildebrandt, Linear programming formulations for attainable region analysis, Chemical Engineering Science 57 (2002) 2015–2028.
[16] K. Klatt, W. Marquardt, Perspectives for process systems engineering – personal views from academia and industry, Computers and Chemical Engineering 33 (2009) 526–550.
[17] Y. Lang, L. Biegler, A software environment for simultaneous dynamic optimization, Computers and Chemical Engineering 32 (2007) 931–942.
[18] P. Lund, An object-oriented environment for process modelling and simulation, PhD thesis, Laboratory of Chemical Engineering, Norwegian Institute of Technology, Trondheim, 1992.
[19] W. Marquardt, Trends in computer-aided process modelling, Computers and Chemical Engineering 20 (6/7) (1996) 591–609.
[20] B. Meyssami, O. Åsbjørnsen, Process modeling – from first principles – method and automation, in: Summer Computer Simulation Conference, 1989, pp. 292–299.
[21] B. Nilsson, Object-oriented modeling of chemical-processes, PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, 1993.
[22] C. Pantelides, SPEEDUP-recent advances in process simulation, Computers and Chemical Engineering 12 (7) (1988) 745–755.
[23] C. Pantelides, P. Barton, Equation-oriented dynamic simulation current status and future perspectives, European Symposium on Computer Aided Process Engineering 2 (1993) S263–S285.
[24] J. Perkins, R. Sargent, SPEEDUP – a computer program for steady-state and dynamic simulations and design of chemical processes, AIChE Symposium Series 214 (1982) 1–11.
[25] Process Systems Enterprise Ltd., gPROMS Advanced User Guide, 2004.
[26] C.F. Sørlie, A computer environment for process modeling, PhD thesis, Laboratory of Chemical Engineering, Norwegian Institute of Technology, Trondheim, 1990.
[27] K. Telnes, Computer-aided-modeling of dynamic processes based on elementary physics, PhD thesis, Division of Engineering Cybernetics, Norwegian Institute of Technology, Trondheim, 1992.
[28] J. Tolsma, P. Barton, On computational differentiation, Computers and Chemical Engineering 22 (4/5) (1998) 475–490.
[29] J. Tolsma, P. Barton, DAEPACK: an open modeling environment for legacy models, Industrial and Engineering Chemistry Research 39 (2000) 1826–1839.
[30] J. Tolsma, J. Clabaugh, P. Barton, Symbolic incorporation of external procedures into process modeling environments, Industrial Engineering and Chemical Research 41 (2002) 3867–3876.
[31] F. Tränkle, A. Gerstlauer, M. Zeitz, E. Gilles, Application of the modeling and simulation environment PROMOT/DIVA to the modeling of distillation processes, Computers and Chemical Engineering Supplement 21 (1997) S841–S846.
[32] F. Tränkle, M. Zeitz, M. Ginkel, E.D. Gilles, PROMOT: a modeling tool for chemical processes, Mathematical and Computer Modelling of Dynamical Systems 6 (2000) 283–307.
[33] J. van de Vusse, Plug-flow type reactor versus tank reactor, Chemical Engineering Science 19 (12) (1964) 994–997.
[34] M. van Goethem, S. Barendregt, J. Grievink, J. Moulijn, P. Verheijen, Towards synthesis of an optimal thermal cracking reactor, Chemical Engineering Research and Design 86 (7) (2008) 703–712.
[35] M. van Goethem, F. Kleinendorst, C. van Leeuwen, N. van Velzen, Equation-based SPYRO model and solver for the simulation of the steam cracking process, Computers and Chemical Engineering 25 (2001) 905–911.
[36] M. van Goethem, F. Kleinendorst, N. van Velzen, M.D.E. Ranzi, Equation based SPYRO model and optimiser for the modelling of the steam cracking process, in: Escape-12 Supplementary Proceedings, 2002.
[37] M. Westerweele, H. Preisig, Concept and design of Modeller, a computer aided modelling tool, Computers and Chemical Engineering Supplement 23 (1999) S751–S754.
[38] E. Woods, The hybrid phenomena theory, PhD thesis, Division of Engineering Cybernetics, Norwegian Institute of Technology, Trondheim, 1993.